USB - unbekannter serieller Bus

Zusammenfassung der vereinfachten Einführung in die Funktionsweise des Universal Serial Bus

Stefan Schürmans <1stein@schuermans.info>

Dezember 2004 (V 1.1)

1 USB - unbekannter serieller Bus

USB steht eigentlich für “Universal Serial Bus” und wurde ca. 1995 von Intel erfunden. Ziel war es, eine neue und einheitliche Schnittstelle zu schaffen, um Peripheriegeräte an den PC anzuschließen, da die betagten Schnittstellen, wie z.B. PS/2-Tastatur, PS/2-Maus, RS232 und Parallelport auf dem ISA-Bus aufbauen und so nicht mehr zur modernen PC-Architektur passen.

Heute ist USB ein weit verbreiteter Standard, der neben den Haupt-Entwicklern HP, Intel, Microsoft und Philips noch von vielen weiteren Unternehmen verwendet wird. Es ist damit zu rechnen, dass gemäß den Plänen von hauptsächlich Intel und Microsoft USB in Zukunft alle älteren Schnittstellen ablösen wird.

2 Eigenschaften von USB

USB bietet als eins der Haupt-Merkmale einheitliche Stecker für alle Geräte. Es gibt einen flachen Stecker (A-Stecker) an der PC-Seite und einen quadratischen Stecker (B-Stecker) für die Verbindung auf der Seite des Geräts (siehe Abbildung 1). Leider ist dieses Konzept mittlwerweile durch eine Vielzahl von Mini-USB-Steckern aufgeweicht worden, da einige Geräte keinen Platz für die Standard-USB-Stecker boten.

Abbildung 1: A-Stecker (links) und B-Stecker (rechts)

Neben der einfachen Punkt-zu-Punkt-Verkabelung eines Geräts zum PC lässt sich die USB-Verkabelung mittels USB-Hubs zu einer Baum-artigen Struktur ausbauen. Dies soll angeblich den typischen Kabelsalat hinter dem PC eindämmen.

Von den Herstellern recht deutlich hervorgehobene Vorteile von USB gegenüber den alten PC-Schnittstellen sind das “hot-plugging”, d.h. das Anschließen von Geräten bei laufendem PC, und die höhere Geschwindigkeit von 1.5 Mbps bei Low-Speed-Geräten oder 12 Mbps bei Full-Speed-Geräten. Mit dem neueren Standard USB 2.0 werden sogar 480 Mbps erreicht.

3 Der USB-Protkoll-Stack

USB verwendet zur Kommunikation zwischen PC und den Geräten ein recht komplexes Protokoll, welches in verschiedenen Schichten aufgebaut ist (siehe Abbildung 2).

Auf der untersten Ebene befindet sich die Hardware mit den elektrischen Spezifikationen für die physikalische Verbindung. Darauf setzt dann die Leitungscodierung auf, die mit Hilfe dieser Verbindung die Übertragung einzelner Bits ermöglicht.

Diesen Dienst benutzt dann die Paket-Schicht, um verschiedene USB-Pakete von und zu den einzelnen Geräten zu übertragen. Mehrere dieser Pakete in unterschiedlichen Richtungen bilden dann die sogenannten USB-Transaktionen. Wiederum mehrere dieser Transaktionen werden dann zu den USB-Transfers zusammengefasst, die dann teilweise direkt von der Applikation genutzt werden.

Spezielle USB-Transfers, die Control-Transfers werden aber noch von USB spezifiziert und dienen der Erkennung und automatischen Konfiguration der eingesteckten Geräte.

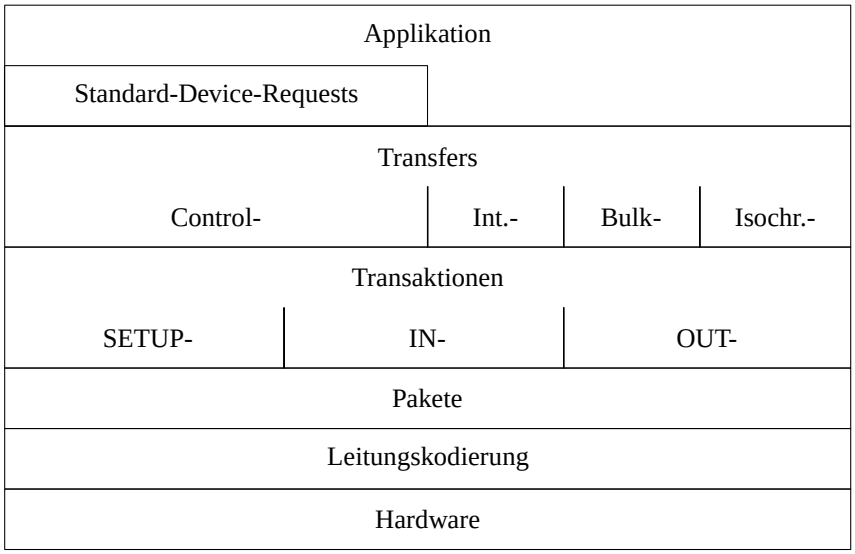

Abbildung 2: USB-Protokoll-Stack

3.1 Hardware

Auf der Hardware-Ebene wird von USB zunächst einmal die Pinbelegung der Stecker festgelegt:

PinNr	Beschreibung	Bezeichnung
1	Versorgungsspannung	+5V
2	negative Datenleitung	D-
3	positive Datenleitung	D+
4	Masse	GND

Die Stromversorgung der Geräte kann also über USB erfolgen. Dabei darf ein nicht konfiguriertes Gerät maximal 100 mA verbrauchen, ein konfiguriertes Gerät nach Absprache mit dem USB-Host, also dem PC, bis zu 500 mA. Im Standby-Modus sind nur 2.5 mA erlaubt, was aber von vielen Geräten nicht eingehalten wird.

Zur Datenübertragung werden vier verschiedene Leitungszustände definiert: Idle, J, K und SE0, die jeweils durch andere Zustände der beiden Datenleitungen D+ und D- darge-

stellt werden. J und K sind normale Zustände, in denen D+ und D- differentiell getrieben werden. Idle ist vom differentiellen Leitungszustand her identisch zu J, aber der Bus ist in diesem Falle passiv und wird nur durch Widerstände in diesen Zustand gebracht. SE0 ist ein besonderer Zustand, in dem beide Datenleitungen aktiv auf Masse getrieben werden. Bestimmte Abfolgen dieser Leitungszustände, die während einer normalen Datenübertragung nicht auftreten, haben eine besondere Bedeutung: “Start of Packet” (SOP) ist z.B. als der Übergang von Idle nach K definiert, “End of Packet” (EOP) ist 2 Takte SE0 gefolgt von mindestens einem Takt Idle. Ein Bus-Reset wird durch mehr als 2.5 μs SE0 ausgelöst.

3.2 Leitungskodierung

Basierend auf den elektrischen Zuständen müssen nun Bytes oder Bits versendet werden. Diese Aufgabe erledigt die Leitungskodierung.

Dazu werden zunächst die zu übertragenden Bytes zu einem Bitstring verkettet (LSB first), in den dann noch vom Bit-Stuffer nach jeweils 6 aufeinanderfolgenden Einsen eine Null eingefügt wird, um nachher auf der Leitung die Synchronisation zu erhalten.

Eine Null wird nämlich als ein Wechsel zwischen J und K kodiert und eine Eins als ein Ausbleiben dieses Wechsels. Daher würde aufgrund leicht abweichender Zeitbasen bei zu vielen Einsen in Folge die Gefahr bestehen, dass die Anzahl der Einsen nicht genau feststellbar ist.

Auf der Empfängerseite wird dieser Prozess in umgekehrter Reihenfolge durchgeführt: Die Wechsel zwischen J und K bzw. das Ausbleiben dieser Wechsel wird wieder in Einsen und Nullen übersetzt. Dann wird vom Bit-Destuffer nach jeweils 6 Einsen die zusätzliche Null entfernt und der Bitstring wird wieder in Bytes übersetzt.

3.3 USB-Pakete

3.3.1 Adressen und Endpoints

Jedes USB-Gerät hat eine eindeutige Adresse von 1 bis 127. Nach dem Einstecken ist diese Adresse zunächst 0 und wird dann durch den Host eingestellt.

Um die eigentliche Kommunikation mit dem Gerät abzuwickeln, werden sogenannte Endpoints verwendet. Dabei handelt es sich um logische Datenquellen und Datensenken. Der Endpoint (EP) mit der Nummer 0 wird von jedem Gerät unterstützt und wird für die Konfiguration verwendet. Die restlichen Endpoints EP1 - EP15 sind von der Anwendung verwendbar.

Zur Erkennung und Konfiguration besitzt jedes Gerät einen Satz Informationen in den sogenannten Descriptors. Diese Informationen werden über EP0 vom Host abgefragt.

3.3.2 USB-Pakete

Auf der Paket-Ebene des USB wird die Adressierung des Geräts und des Endpoints realisiert. Dazu enthalten einige Pakete neben einer Synchronisierung und dem Pakettyp die Geräte-Adresse und die Endpoint-Nummer. Die wichtigsten dieser Pakete sind SETUP (siehe Abbildung 3), IN und OUT. Diese Pakete initiieren eine Transaktion und legen dabei das Gerät und den Endpoint für diese Transaktion fest.

SYNC	SETUP	ADDR	EP	CRC5	EOP
00000001	0x2D	0x01	0x00	0x17	EOP

Abbildung 3: SETUP-Paket

Die nächste Gruppe von Paketen sind DATA0 und DATA1. Sie beziehen sich immer auf die aktuelle Transaktion und enthalten daher keine Adressierungs-Informationen, sondern

nur Nutzdaten. Dabei werden die beiden Pakettypen immer abwechselnd gesendet, um bei verlorenen Quittungspaketen eine doppelte Datenauslieferung zu verhindern.

Die Quittungspakete ACK, NAK und STALL sind die letzte wichtige Gruppe von USB-Paketen. ACK wird zur Bestätigung eines erfolgreichen Empfangs verwendet, NAK bei nicht erfolgreichem Empfang oder falls keine Daten zur Sendung bereit liegen. STALL zeigt einen Fehler auf dem aktuellen Endpoint an. Diese Pakete enthalten weder Adressierungs-Informationen noch Nutzdaten.

3.4 USB-Transaktionen

Die USB-Transaktionen dienen nun wirklich zur Übertragung von Daten an ein oder von einem bestimmten Gerät, dabei werden Setup-Daten an ein Gerät ausdrücklich von normalen Daten unterschieden.

Um Setup-Daten an ein Gerät zu senden, überträgt der Host ein SETUP-Paket, gefolgt von einem DATAx-Paket. Dieses Paket wird bei Erfolg vom Gerät mit einem ACK-, ansonsten mit einem NAK-Paket beantwortet. Diesen Vorgang bezeichnet man als SETUP-Transaktion.

Der Empfang von Daten von einem Gerät geschieht mit einer IN-Transaktion. Dazu sendet der Host ein IN-Paket. Das Gerät antwortet mit einem DATAx-Paket, welches die Nutzdaten enthält und vom Host wieder mit einem ACK-Paket bestätigt wird. Hat das Gerät keine Nutzdaten, so antwortet es mit einem NAK-Paket anstelle des DATAx-Pakets.

Eine OUT-Transaktion beginnt mit einem vom Host gesendeten OUT-Paket. Es folgt ein DATAx-Paket mit den Nutzdaten. Das Gerät antwortet wie bei der SETUP-Transaktion mit einem ACK-Paket oder einem NAK-Paket. Es wird auch mit einem NAK-Paket geantwortet, falls gerade kein Pufferspeicher für die Daten bereit steht.

Als Sonderfall kann ein Gerät bei einer IN- oder OUT-Transkation auch mit einem STALL-Paket antworten. Dies zeigt einen schweren Fehler an, der nur durch einen oder mehrere Konfigurationsschritte durch den Host behoben werden kann.

Wie man sieht, ist kein Gerät berechtigt eine Transaktion zu initiieren. Dies ist ein wesentliches Konzept auf dem USB. Nur der Host kann Transaktionen starten und muss alle Geräte periodisch abfragen, falls er Daten erwartet.

3.5 USB-Transfers

Es gibt vier Typen von Transfers im USB-Protokoll: Control-, Interrupt-, Bulk- und Isochronous-Transfers. Control-Transfers laufen nur auf Endpoint 0 ab, die drei anderen Transfer-Arten verwenden die Endpoints 1 - 15. Bulk- und Isochronous-Transfers stehen bei Low-Speed-Geräten nicht zur Verfügung.

Die Erkennung und Konfiguration eines Geräts benutzt Control-Transfers. Diese können insgesamt bis zu 10% der Bandbreite nutzen und ihre Pakete werden bei Übertragungsfehlern wiederholt.

Die Übertragung von Konfigurations-Daten zu einem Gerät besteht aus einer SETUP-Transaktion, beliebig vielen OUT-Transaktionen und einer abschließenden IN-Transaktion mit 0 Byte Daten. Das Auslesen von Konfigurations-Daten eines Geräts besteht aus einer SETUP-Transaktion, mindestens einer IN-Transaktion und einer abschließenden OUT-Transaktion mit 0 Byte Daten.

Interrupt-Transfers hören sich zunächst so an, als ob hier ein Gerät eigenständig den Host über ein Ereignis informieren würde. Dies ist aber nicht der Fall. Sie werden lediglich dort eingesetzt, wo man normalerweise einen Interrupt verwenden würde. Es handelt sich hier auch um ein Host-gesteuertes Polling, welches z.B. für Statusabfragen oder die Übertragung kleiner Datenmengen eingesetzt wird.

Der Host initiiert pro Zeitttakt (1ms, 2ms, 4ms, ..., 128ms) eine IN- oder OUT-Transkation um Daten zu übertragen. Dabei werden Pakete bei Übertragungsfehlern wiederholt und es

können bis zu 90% der Bandbreite (gemeinsam mit Isochronous-Transfers) genutzt werden.

Bulk-Transfers werden zur Übertragung großer und zeitunkritischer Datenmengen eingesetzt. Es können beliebig viele IN- oder OUT-Transaktionen zu jedem Zeitpunkt vom Host gestartet werden und bei Übertragungsfehlern wird eine erneute Übertragung veranlasst. Allerdings gibt es keine reservierte Bandbreite für BULK-Transfers, es kann nur die freie Bandbreite genutzt werden.

Isochronous-Transfers sind für den zeitkristischen Datenaustausch mit konstanter Bandbreite und ohne Fehlerschutz vorgesehen. Es werden pro USB-Zeittakt (1 ms) immer gleich viele IN- bzw. OUT-Transaktionen ausgeführt. Dazu kann (zusammen mit Interrupt-Transfers) bis zu 90% der Bandbreite benutzt werden.

3.6 USB-Standard-Device-Requests

3.6.1 USB-Descriptors

Die Informationen über ein Gerät werden im Gerät in den sogennanten Descriptors gespeichert. Dazu enthält jedes Gerät einen Device-Descriptor, der u.a. eine eindeutige Hersteller- und Geräte- Nummer enthält. Dies wird z.B. dazu genutzt, den richtigen Treiber für ein eingestecktes USB-Gerät zu laden. Weiter enthält der Device-Descriptor eine Kennnummer der Geräte-Klasse, so dass z.B. eine USB-Maus mit dem Standard-USB-Maustreiber benutzt werden kann, wenn auch nicht mit allen Sonderfunktionen.

In den Configurations-Descriptors werden die verschiedenen möglichen Konfigurationen eines Geräts abgelegt. Somit ist es möglich, dass z.B. eine Webcam in einer Konfiguration nur ein Bild liefert und in einer anderen Konfiguration Bild und Ton.

Jede Konfiguration kann mehrere Interfaces besitzen, die in den Interface-Descriptors beschrieben sind. So würde z.B. die Webcam in der zweiten Konfiguration ein Interface für das Bild und ein Interface für den Ton enthalten.

Um diese Einstellungsvielfalt noch zu übertreffen, kann jedes Interface noch verschiedene Einstellungen (Alternate Settings) unterstützen. Ein Beispiel ist hier eine Audio-Übertragung mit unterschiedlicher Datenrate. Für diese Einstellungen gibt es aber keine eigenen Descriptoren, es werden einfach verschiedene Interface-Descriptoren für das gleiche Interface angegeben.

Die Endpoints eines Interfaces einer Konfiguration, werden durch die Endpoint-Descriptors beschrieben, die neben der Endpoint-Nummer den Transfer-Typ und die Transfer-Richtung enthalten.

Um bei diesen vielfältigen Konfigurationen und Einstellungen den Durchblick zu behalten, kann mit Hilfe von String-Descriptoren Klartext zu den einzelnen Kenndaten angegeben werden. So kann z.B. dem Benutzer beim Einstecken eines USB-Geräts nicht die Hersteller- und Geräte-Nummer, sondern der Hersteller- und Geräte-Name angezeigt werden.

Der Zusamenhang zwischen den einzelnen Descriptoren ist in Abbildung 4 skizziert.

3.6.2 USB-Standard-Device-Requests

Über EP 0 werden mit Hilfe von SETUP-Transkationen diese Descriptors abgefragt und das Gerät konfiguriert. Dazu sind die Standard-Device-Requests definiert. Hier werden nur einige grob skizziert, da dies sonst den Umfang dieser Zusammenfassung sprengen würde.

SetAddress ist z.B. ein Control-Transfer, der die Adresse des USB-Geräts setzt. Dies wird meist einmalig nach dem Einstecken des Geräts durchgeführt.

Ein weiterer wichtiger Control-Transfer ist GetDescriptor, der einen Descriptor vom Gerät abfragt. Damit kann z.B. nach dem Einstecken eines Geräts festgestellt werden, um welches Gerät es sich handelt, welcher Treiber geladen werden muss und welche Resourcen auf dem USB reserviert werden müssen.

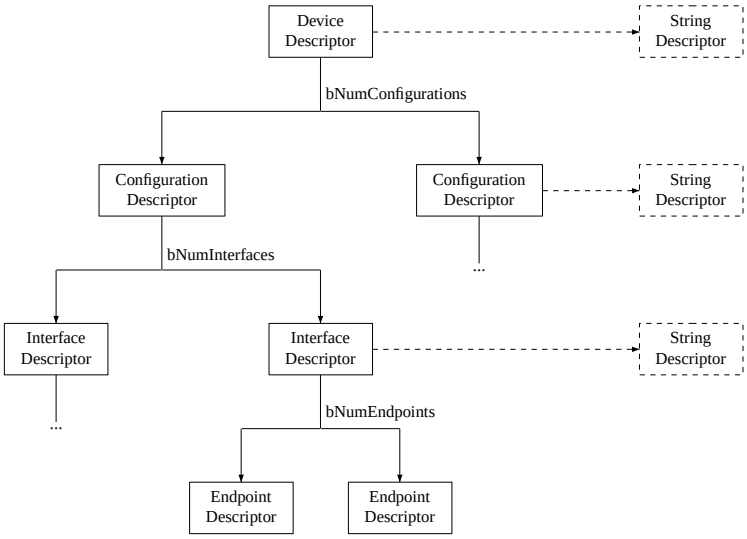

Abbildung 4: Zusammenhang zwischen den USB-Descriptoren

Schließlich kommt noch SetConfiguration zum Einsatz, womit die aktuelle Konfiguration des Geräts gesetzt wird und das Gerät aktiviert wird. Nach diesem Vorgang kann der Treiber oder auch eine Applikation über die Endpoints 1 - 15 mit Hilfer von Interrupt-, Bulk- und Isochronous-Transfers mit dem Gerät kommunizieren.

Eine interessante Tatsache ist, dass die Abfrage der Descriptoren und das Setzen der Adresse auf unterschiedlichen Betriebssystemen in anderer Reihenfolge abläuft. So fragt z.B. Linux die Descriptors erst nach dem Setzen der Adresse ab, während Windows den Device-Descriptor bereits vorher abfragt. Mit dieser Information könnte man ein USB-Gerät bauen, welches nur unter bestimmten Betriebssystemen funktioniert.

3.7 USB-PowerOn-Enumeration

Ein Problem ist die Verteilung der Adressen an alle eingesteckten Geräte beim Einschalten des Rechners. Alle Geräte haben zu diesem Zeitpunkt die Adresse 0. Ein SetAdress an Gerät 0 würde die Adresse aller Geräte ändern.

Die Lösung liegt im Verhalten der Hubs. Diese haben nach dem Einschalten ihre Ports deaktiviert. So kann der Host mit SetAdress die Adresse des ersten Hubs auf 1 ändern und dann den ersten Port aktivieren. Das dort angeschlossene Gerät erhält die Adresse 2. So werden nach und nach alle Hub-Ports aktiviert und jedes Gerät erhält eine eindeutige Adresse.

4 USB-Geräte-Hardware

Wie man USB-Geräte selbst bauen kann, soll nun auch noch angesprochen werden. Das wesentliche Problem ist die hohe Datenrate auf dem USB, was eine Software-Implementierung mit einem Mikrocontroller allenfalls noch für Low-Speed erlaubt. Aber das ist auch nicht notwendig, weil es für diese Aufgabe Hardwarelösungen gibt.

Zum einen gibt es USB-Interface-Bausteine, die nur das USB-Protokoll in Hardware implementieren und die Daten in FIFO-Puffern bereitstellen. Auf der anderen Seite bieten sie meist ein Standard-Microcontroller-Interface an.

Die andere Klasse von USB-Hardware sind sogenannte USB-Controller. Dabei handelt es sich um Mikrocontroller, die ein eingebautes USB-Interface besitzen. Ein Beispiel ist die EZ-USB-Serie von Cypress.

Hier soll nun kurz der USB-Interface-Baustein USBN9604 von National Semiconductor vorgestellt werden. Dieser Baustein unterstützt USB 1.1 Full-Speed und ist mit 28 Pins im Abstand von 1.27 mm noch ohne weiteres von Hand lötbar. Durch seinen integrierten 3.3 V Spannungsregler wird die Beschaltung auf USB-Seite sehr vereinfacht. Auch der Preis ist mit 5 Euro bei 1 Stück durchaus noch akzeptabel.

Im einfachsten (und schnellsten) Fall wird der USBN9604 über einen 8 Bit breiten kombinierten Daten-/Adress-Bus an einen Mikrocontroller angeschlossen (siehe Abbildung 5). So lassen sich dann die 64 internen Register lesen und beschreiben.

Mit Hilfe der Register ist dann die Konfiguration des Chips und Zugriff auf die FIFO-Puffer vom Microcontroller aus möglich. Die Protokolle bis unterhalb der Transfer-Ebene sind im USBN9604 in Hardware implementiert, so dass man sich im wesentlichen in der Software nur noch um die Standard-Device-Requests und die eigentliche Nutzdaten-Kommunikation kümmern muss.

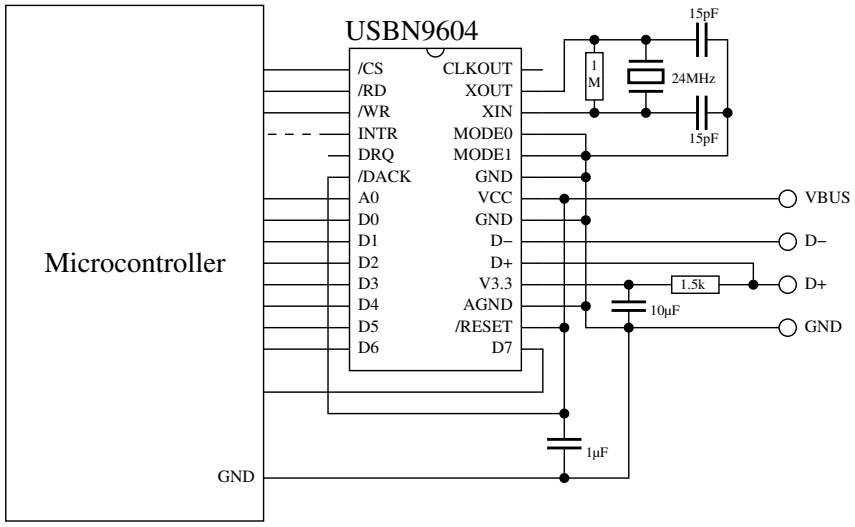

Abbildung 5: Prinzip-Schaltung USBN9604 an Microcontroller

5 weitere Informationen

- http[s]://www.usb.org/
  - u.a. USB-Spezifikation
- USBN9604 Datenblatt
- /usr/src/linux/drivers/usb/usb-skeleton.c
- http[s]://1stein.schuermans.info/hackerport/
  - Beispiel eines selbstgebauten USB-Geräts